Introduction to FluxML (+ Julia)

Data Umbrella Webinar

Kyle Daruwalla (14 June 2023)

Julia programming language

How is it different?

- Dynamic programming language
- Familiar syntax for Matlab, Python, and Numpy users
- Ahead-of-time (AoT) compilation
- Designed for scientific computing

Why choose Julia for ML?

- Sensible package manager (Pkg.jl)
  - Binaries/artifacts shipped and version controlled by package manager
  - Reproducible environments
- Multiple dispatch + abstract array interface
  - Easily write code that specializes for different hardware devices
- Meta-programming and code reflection for automatic differentiation
- REPL-based development for rapid prototyping

Getting familiar with Julia

Preliminaries

https://docs.julialang.org/en/v1/stdlib/REPL/

Baby's first array (tensor)

```
julia> x = ones(2, 3)
2×3 Matrix{Float64}:
 1.0  1.0  1.0
 1.0  1.0  1.0

julia> y = 2 * ones(3)
3-element Vector{Float64}:
 2.0
 2.0
 2.0
```

```
julia> x * y
2-element Vector{Float64}:
 6.0
 6.0

julia> transpose(y)
1×3 transpose(::Vector{Float64}) with eltype Float64:
 2.0  2.0  2.0
```

```
julia> ndims(x)
2

julia> ndims(y)
1

julia> typeof(x)
Matrix{Float64} (alias for Array{Float64, 2})

julia> typeof(y)
Vector{Float64} (alias for Array{Float64, 1})
```

```
julia> x .+ transpose(y)
2×3 Matrix{Float64}:
 3.0  3.0  3.0
 3.0  3.0  3.0
```

Multiple dispatch

```
julia> f(x) = 2 * x
f (generic function with 1 method)

julia> f(2)
4
```

```
julia> f(x::Int) = sqrt(x)
f (generic function with 2 methods)

julia> f(2)
1.4142135623730951

julia> f(2.0)
4.0
```

```
julia> methods(f)
# 2 methods for generic function "f" from Main:
 [1] f(x::Int64)
     @ REPL[3]:1
 [2] f(x)
     @ REPL[1]:1
```

Multiple dispatch + array interfaces

```
julia> using OneHotArrays

julia> indices = rand(0:5, 4)
4-element Vector{Int64}:
 3
 2
 3
 2

julia> oh = onehotbatch(indices, 0:5)
6×4 OneHotMatrix(::Vector{UInt32}) with eltype Bool:
 ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅
 ⋅  1  ⋅  1
 1  ⋅  1  ⋅
 ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅
```

```
julia> embedding = rand(3, 6)
3×6 Matrix{Float64}:
 0.488217  0.0220455  0.290054   0.185517  0.687396  0.698643
 0.309948  0.935395   0.0736855  0.293578  0.555564  0.168734
 0.749137  0.820596   0.837678   0.943308  0.797423  0.757755

julia> embedding * oh
3×4 Matrix{Float64}:
 0.185517  0.290054   0.185517  0.290054
 0.293578  0.0736855  0.293578  0.0736855
 0.943308  0.837678   0.943308  0.837678

julia> @which embedding * oh
*(A::AbstractMatrix, B::Union{OneHotArray{var"#s13", 1, var"N+1", I}, Base.R
+1", <:OneHotArray{var"#s13", <:Any, <:Any, I}}} where {var"#s13", var"N+1",
     @ OneHotArrays ~/.julia/packages/OneHotArrays/T3yiq/src/linalg.jl:7
```

Understandable GPU programming

```
julia> using CUDA

julia> x = rand(3, 4)
3×4 Matrix{Float64}:
 0.289258  0.0165733  0.151748    0.0634891
 0.913348  0.437743   0.17894     0.84348
 0.899995  0.769801   0.00969918  0.605775

julia> y = cu(x)
3×4 CuArray{Float32, 2, CUDA.Mem.DeviceBuffer}:
 0.289258  0.0165733  0.151748    0.0634891
 0.913348  0.437743   0.17894     0.84348
 0.899995  0.769801   0.00969918  0.605775
```

https://cuda.juliagpu.org/stable/

```
julia> f(x) = 2 * x
f (generic function with 1 method)

julia> f(x::AbstractArray) = sqrt.(x)
f (generic function with 2 methods)

julia> f(x::CuArray) = x .^ 2
f (generic function with 3 methods)

julia> f(4)
8

julia> f(x)
3×4 Matrix{Float64}:
 0.537827  0.128737  0.389548   0.25197
 0.955692  0.661621  0.423013   0.918411
 0.948681  0.877383  0.0984844  0.778315

julia> f(y)
3×4 CuArray{Float32, 2, CUDA.Mem.DeviceBuffer}:
 0.0836703  0.000274673  0.0230273   0.00403087
 0.834204   0.191619     0.0320196   0.711458
 0.809992   0.592594     9.40741f-5  0.366963
```

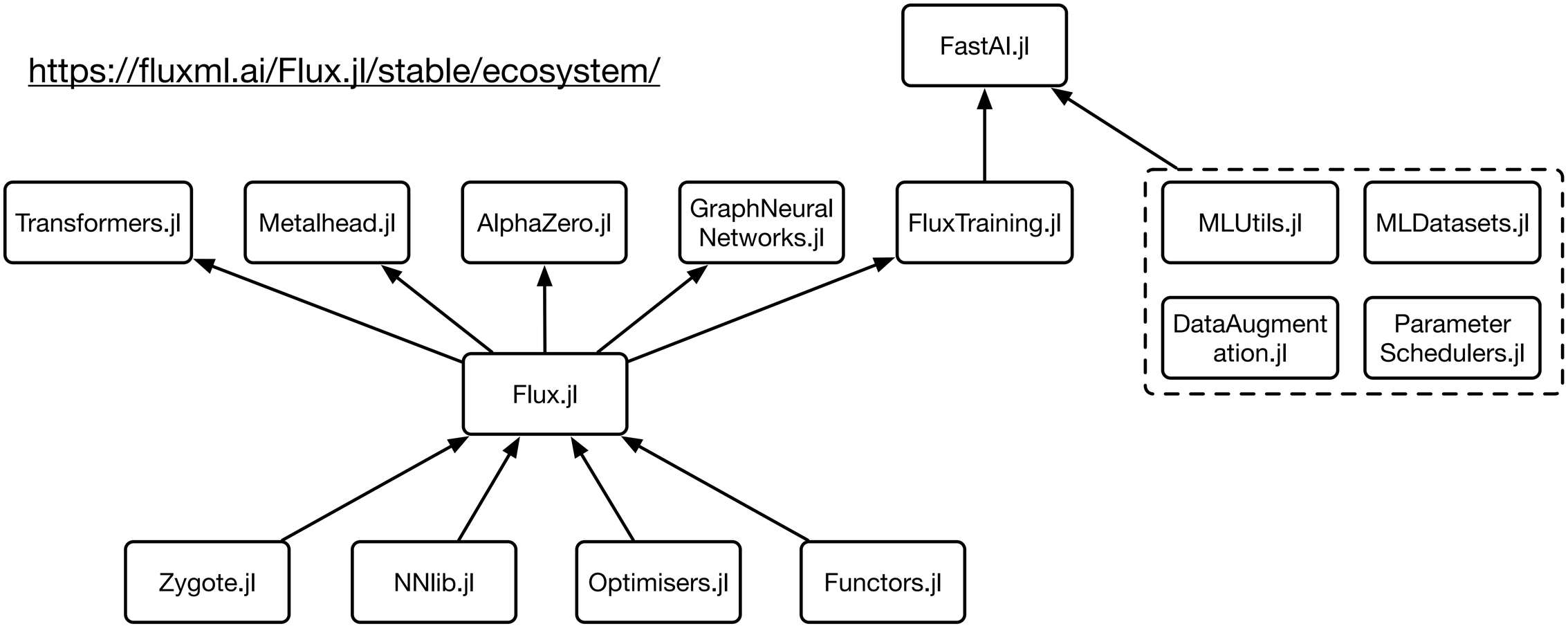
FluxML Ecosystem

FluxML org


fluxml.ai
flux
Documentation
Model Zoo
GitHub
Ecosystem
Blog
GSoC
Governance
Contribute
flux
The Elegant Machine Learning Stack
Flux is a 100% pure-Julia stack and provides lightweight abstractions on top of Julia's native GPU and AD support. It makes the easy things easy while remaining fully hackable.
Try It Out
GitHub
Follow on Twitter

FluxML ecosystem

https://fluxml.ai/Flux.jl/stable/ecosystem/



Linear regression in Flux.jl

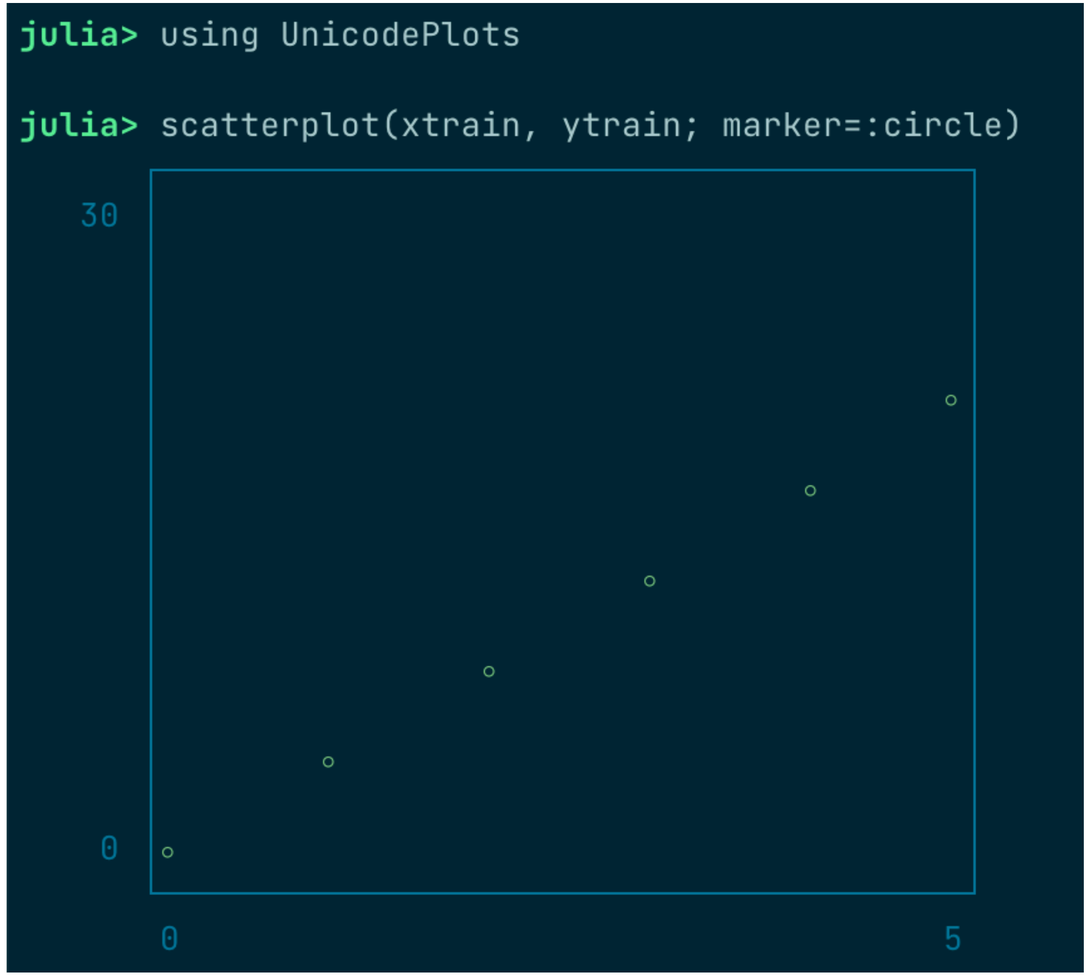
Defining the dataset

Ground Truth Model

$$y = wx + b$$

```
julia> using Flux

julia> true_model(x) = 4 * x + 2
true_model (generic function with 1 method)

julia> xtrain, xtest = collect(0:5), collect(6:10)
([0, 1, 2, 3, 4, 5], [6, 7, 8, 9, 10])

julia> ytrain, ytest = true_model.(xtrain), true_model.(xtest)
([2, 6, 10, 14, 18, 22], [26, 30, 34, 38, 42])
```

```
julia> using UnicodePlots

julia> scatterplot(xtrain, ytrain; marker=:circle)
```



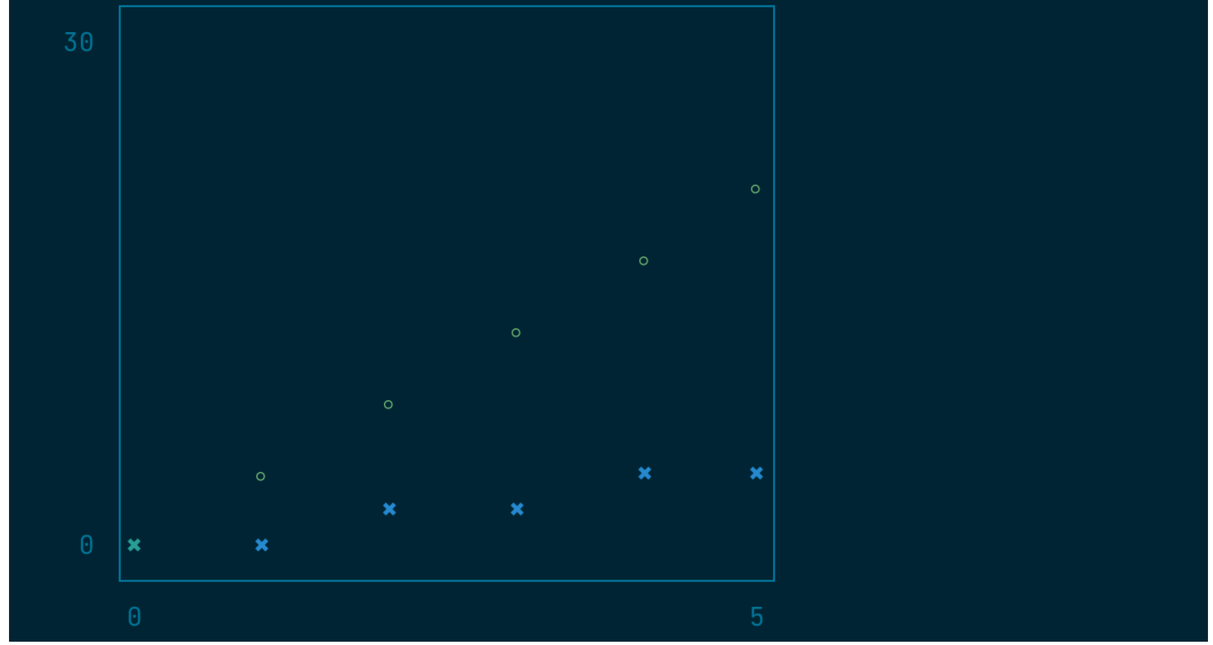
https://fluxml.ai/Flux.jl/stable/models/overview/

Linear regression in Flux.jl

Defining a model

```
julia> model(w, b, x) = w * x + b
model (generic function with 1 method)

julia> w0, b0 = rand(), rand()
(0.9335443346723611, 0.3303649518307457)
```

```
julia> fig = scatterplot(xtrain, ytrain; marker = :circle);

julia> scatterplot!(fig, xtrain, model.(w0, b0, xtrain); marker = :xcross)
```



https://fluxml.ai/Flux.jl/stable/models/overview/

Linear regression in Flux.jl

Taking a gradient

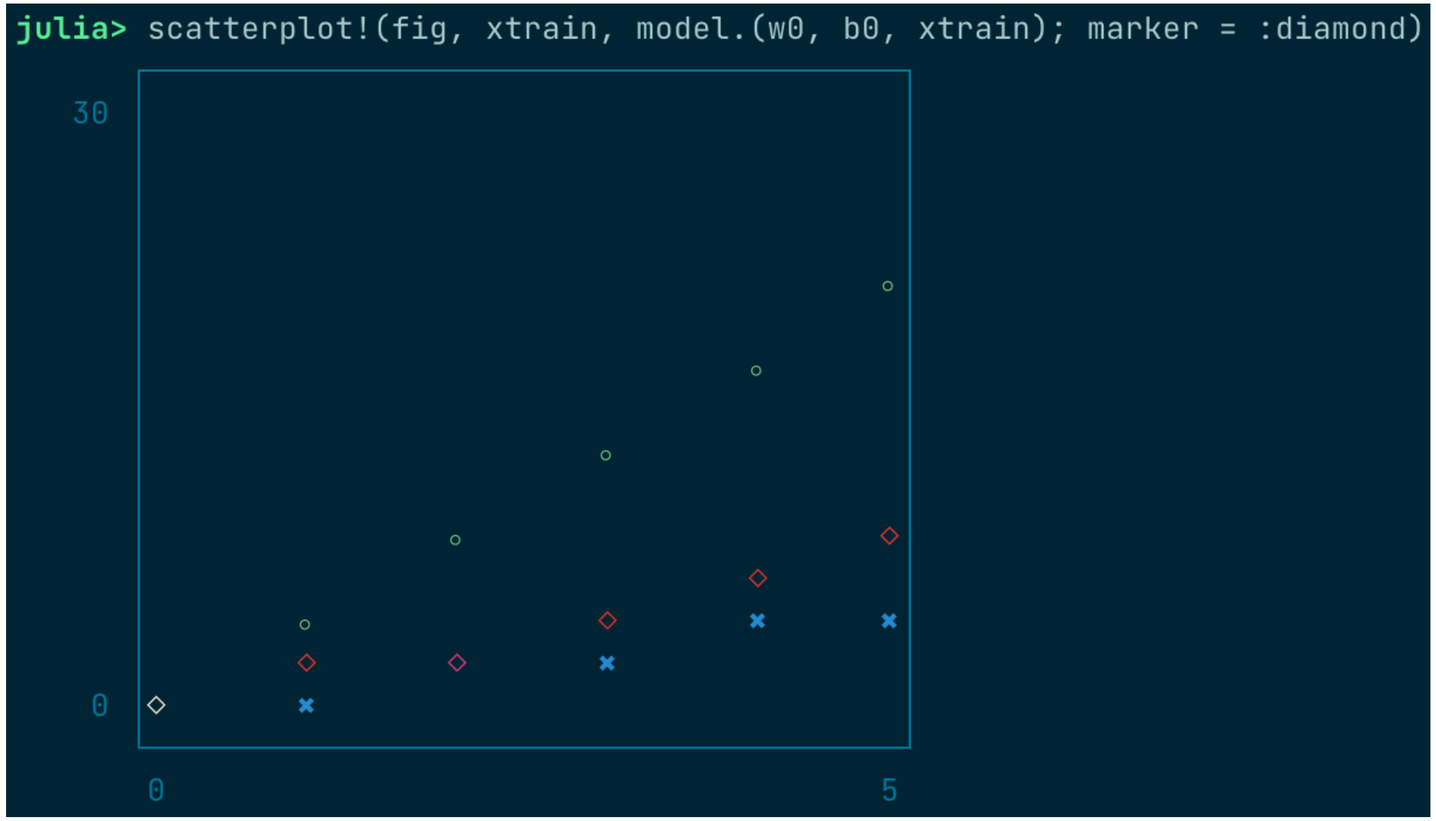
```
julia> dw, db = Flux.gradient(w0, b0) do w, b
           y = model.(w, b, xtrain)
           return Flux.Losses.mse(y, ytrain)
       end
(-64.5665291051863, -18.6715484229767)
```

https://fluxml.ai/Flux.jl/stable/models/overview/

Linear regression in Flux.jl

Updating the parameters

```
julia> w0 = w0 - 0.01 * dw
1.5792096257242224

julia> b0 = b0 - 0.01 * db
0.5170804360605128
```



https://fluxml.ai/Flux.jl/stable/models/overview/

Layers in Flux.jl

```
julia> struct Linear{T, S}
           w::T
           b::S
       end

julia> Flux.@functor Linear
```

```
julia> (m::Linear)(x) = m.w * x + m.b
```

Layers in Flux.jl

```
julia> linear_model = Linear(w0, b0)
Linear{Float64, Float64}(1.579209625724224, 0.5170804360605128)
```

```
julia> dm = Flux.gradient(linear_model) do m
           y = m.(xtrain)
           return Flux.Losses.mse(y, ytrain)
       end
((w = -51.79575468141999, b = -15.069790999257854),)
```

A more complex example

Loading MNIST dataset

```
julia> using MLDatasets

julia> train_data = MLDatasets.MNIST()
dataset MNIST:
  metadata  =>    Dict{String, Any} with 3 entries
  split     =>    :train
  features  =>    28×28×60000 Array{Float32, 3}
  targets   =>    60000-element Vector{Int64}

julia> test_data = MLDatasets.MNIST(split=:test)
dataset MNIST:
  metadata  =>    Dict{String, Any} with 3 entries
  split     =>    :test
  features  =>    28×28×10000 Array{Float32, 3}
  targets   =>    10000-element Vector{Int64}
```

https://github.com/FluxML/model-zoo/blob/master/vision/mlp_mnist/mlp_mnist.jl

Loading MNIST dataset

```
julia> function build_dataloader(data::MNIST; batchsize = 64)
           xflat = reshape(data.features, 28*28, :)
           yhot = Flux.onehotbatch(data.targets, 0:9)

           return Flux.DataLoader((xflat, yhot); batchsize, shuffle = true)
       end
build_dataloader (generic function with 1 method)

julia> train_loader = build_dataloader(train_data)
938-element DataLoader(::Tuple{Matrix{Float32}, OneHotMatrix{UInt32, Vector{UInt32}}}, shuffle=true,
 batchsize=64)
  with first element:
  (784×64 Matrix{Float32}, 10×64 OneHotMatrix(::Vector{UInt32}) with eltype Bool,)
```

https://github.com/FluxML/model-zoo/blob/master/vision/mlp_mnist/mlp_mnist.jl

Defining the model and loss function

```
julia> m = Chain(Dense(28*28 => 64, relu), Dense(64 => 10))
Chain(
  Dense(784 => 64, relu),               # 50_240 parameters
  Dense(64 => 10),                      # 650 parameters
)                   # Total: 4 arrays, 50_890 parameters, 199.039 KiB.

julia> loss(m, x, y) = Flux.Losses.logitcrossentropy(m(x), y)
loss (generic function with 1 method)

julia> loss(m, first(train_loader)...)
2.2980614f0
```

https://github.com/FluxML/model-zoo/blob/master/vision/mlp_mnist/mlp_mnist.jl

Defining the accuracy metric

```
julia> accuracy(m, x, y) = mean(Flux.onecold(m(x)) .== Flux.onecold(y))
accuracy (generic function with 1 method)

julia> accuracy(m, data) = mean(accuracy(m, x, y) for (x, y) in data)
accuracy (generic function with 2 methods)

julia> test_loader = build_dataloader(test_data);

julia> accuracy(m, test_loader)
0.07285031847133758
```

https://github.com/FluxML/model-zoo/blob/master/vision/mlp_mnist/mlp_mnist.jl

Writing the training loop

```
julia> opt_state = Flux.setup(Adam(3e-4), m);
```

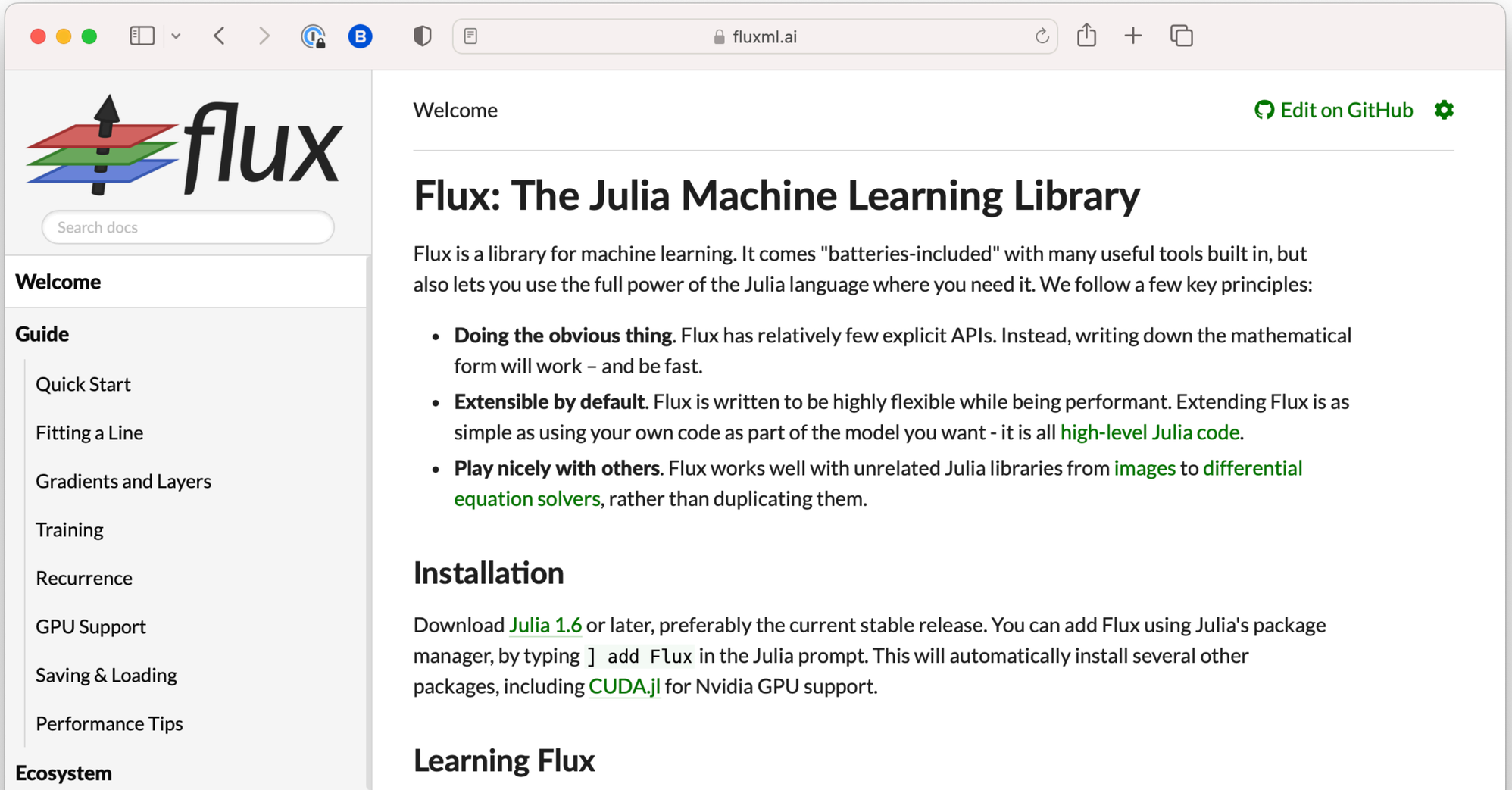
```
julia> for epoch in 1:30
           total_loss = 0.0
           for (x, y) in train_loader
               # Compute loss + gradients
               loss_batch, grads = Flux.withgradient(_m -> loss(_m, x, y), m)
               # Update parameters + optimizer
               opt_state, m = Flux.update!(opt_state, m, grads[1])
               # Accumulate epoch loss
               total_loss += loss_batch
           end
           avg_loss = total_loss / length(train_loader)
           acc = accuracy(m, test_loader)
           @info "Epoch = $epoch" avg_loss acc
       end
┌ Info: Epoch = 1
│   avg_loss = 0.2641783334902609
└   acc = 0.9324243630573248
┌ Info: Epoch = 2
│   avg_loss = 0.21641308431829342
└   acc = 0.9435708598726115
┌ Info: Epoch = 3
```

https://github.com/FluxML/model-zoo/blob/master/vision/mlp_mnist/mlp_mnist.jl

Next steps

Visit our documentation for more!

https://fluxml.ai/Flux.jl/stable/



Become a contributor!

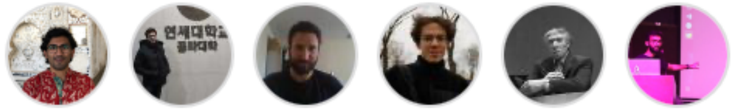
https://github.com/FluxML/Flux.jl/blob/master/CONTRIBUTING.md

Get help

The Flux community is more than happy to help you with any questions that you might have related to your contribution. You can get in touch with the community in any of the following channels:

- **Discourse forum:** Go to Machine Learning in Julia community
- **Community team:** Join the group of community maintainers supporting the FluxML ecosystem (see the Zulip stream as well)
- **Slack:** Join the Official Julia Slack for casual conversation (see `#flux-bridged` and `#machine-learning`)
- **Zulip:** Join the Zulip Server for the Julia programming language community
- **Stack Overflow:** Search for Flux/ ML tags
- **Events:** Attend the ML bi-weekly calls

Contributors This could be you!



darsnack, mcognetta, and 4 other contributors

Attend JuliaCon 2023

https://juliacon.org/2023/